The VMware Mobile Virtualization Platform: is that a hypervisor in your pocket?

Ken Barr Prashanth Bungale Stephen Deasy Viktor Gyuris Perry Hung
Craig Newell Harvey Tuch Bruno Zoppis
{kbarr, prash, sdeasy, gyuris, perry, craign, htuch, bzoppis}@vmware.com
VMware, Inc.

ABSTRACT

The virtualization of mobile devices such as smartphones, tablets, netbooks, and MIDs offers significant potential in addressing the mobile manageability, security, cost, compliance, application development and deployment challenges that exist in the enterprise today. Advances in mobile processor performance, memory and storage capacities have led to the availability of many of the virtualization techniques that have previously been applied in the desktop and server domains. Leveraging these opportunities, VMware's Mobile Virtualization Platform (MVP) makes use of system virtualization to deliver an end-to-end solution for facilitating employee-owned mobile phones in the enterprise. In this paper we describe the use case behind MVP, and provide an overview of the hypervisor's design and implementation. We present a novel system architecture for mobile virtualization and describe key aspects of both core and platform virtualization on mobile devices.

1. INTRODUCTION

Over the past decade the mobile phone has evolved from a voice-centric device to a mobile personal computer. We can observe multiple progressive generations in the enterprise adoption of such devices. There has been a shift from pure voice support to email and web browsing, with current trends towards the mimicking of desktop functionality. On the horizon we expect mobile devices to become true first-class enterprise endpoints with rich applications and core enterprise connectivity.

As mobile computing advances, it brings some familiar challenges to enterprises; application management and interoperability, security and manageability, consumerization of IT and persona management. Independent research firm, Forrester Research, Inc. found in 2008 that firms expected a 3× increase in enterprise employee smartphone users by 2013 [16]. This will lead to an acceleration of the rate at which these challenges become IT concerns. In addition, the diversity of usage and maturation of enterprise applications on mobile platforms will limit the scalability of previous ad hoc solutions, for example pushing services like email outside of enterprise firewalls.

From an end-user perspective there is a healthy desire to consolidate personal physical device ownership and the dilemma of carrying a smartphone for work is no different. Rather than carrying two devices, allowing enterprise IT stewardship of a personal device or even sacrificing the personal device for some enterprise-provided device, we feel that virtualization offers an opportunity to provide the best solution by preserving isolation of environments without requiring a second physical device.

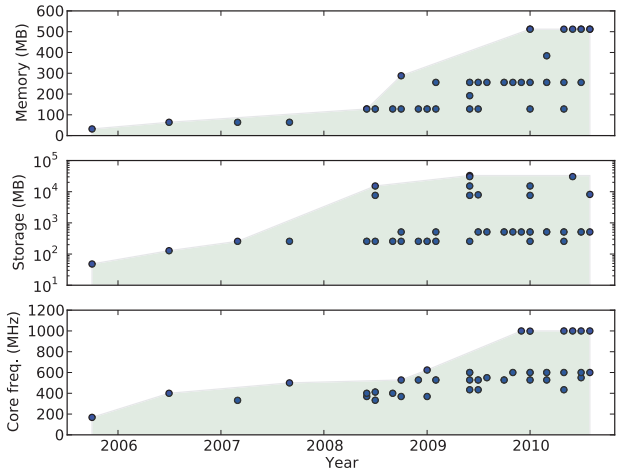


Figure 1: Memory capacity, internal storage and clock frequency for a sample of smartphones [25].

Co-existing virtual phones on a device represents a very attractive alternative to existing solutions for these problems. The rapid pace of hardware advances in mobile devices over the past several years, as shown in Figure 1, has led to a class of devices with resources capable of supporting multiple virtual phones where the virtualization overhead is small.

The VMware Mobile Virtualization Platform (MVP) is an end-to-end solution for enterprise management of employee owned phones, encompassing a hypervisor for ARM-based devices, an enterprise virtual phone and remote device management. In this paper we focus on the hypervisor layer and provide the following contributions:

- A system architecture for a Type 2 mobile hypervisor, a novel design point for a mobile hypervisor with significant device compatibility and deployment advantages.
- A lightweight paravirtualization technique for ARMv7 cores aimed at minimizing the total system complexity.

- Device and platform virtualization approaches for storage, networking and telephony, the key devices in enabling the performance, reliability and security of the MVP solution.
- An application of the MVP hypervisor to the virtualization of the Android [18] operating system. Android is the fastest growing mobile operating system today [17] and will be used as a running example for both guest and host.

The following section provides some discussion on the design goals that fall out of the above motivating use case. The MVP system architecture and details on the virtualization approaches adopted for the core and devices are given in Section 3, the user interface of a virtualized Android-based system is described in Section 4 and we conclude with related work and MVP's status. We limit the scope of discussion to smartphones but note that the emerging class of ARM-based tablet and netbook devices share with smartphones many hardware and software characteristics and could be substituted in the rest of the paper.

2. DESIGN GOALS

The MVP hypervisor enables an enterprise provisioned and managed virtual phone to execute on a personal phone owned by an employee of the organization, a Bring Your Own Device (BYOD) model. It provides a second completely encapsulated smartphone stack by virtualizing the system at the hardware level and providing a virtual machine (VM) platform for the enterprise stack to execute on. We describe below the design goals that informed the hypervisor's technical architecture:

- **Portability**. Since the employee's personal phone is the physical platform, it is important to have the MVP hypervisor able to run on a wide variety of possible mobile devices.

  Figure 2 provides a block level diagram of an example smartphone. It is a simplified view, since some of the components such as the GSM stack may have additional processor cores and memory devices, but even at this level of detail it's evident that a smartphone platform consists of a significant number of diverse components. Unlike the desktop and server domains, most components are integrated on a phone's System-on-Chip (SoC). A SoC is made up of a number of IP blocks, including the processor core. For each version of the standardized ARM architecture there are a number of cores available from ARM and architectural licensees, each with implementation defined behaviors within that allowed by the architecture specification. Beyond the core there is a plethora of device components, each with its own set of interfaces and programming model.

  To maximize the number of devices a mobile hypervisor executes on it should avoid undue dependency on the many and varied specifics of a phone's hardware. This implies a rethink of mobile hypervisor system structure which we describe in Section 3.

- **Compatibility**. An effective solution to enterprise application development and management requirements needs to present a single virtual platform as a target for enterprise applications. The role of the virtualization layer is to map this abstraction to the underlying phone hardware and system software where the VM is deployed.

  Most applications are written to specific mobile middleware and operating systems, for example Android, iOS or Symbian. It is common for them to have dependencies on specific versions of these environments. By taking an existing set of components as the software base of the virtual platform, such as the Android libraries, services and kernel, it's possible to facilitate the reuse of existing enterprise applications in the virtual phone environment. In addition this allows the enterprise to tap into the existing ecosystem and developer base surrounding the platform. The hypervisor is concerned with providing a virtual hardware abstraction capable of supporting efficiently the execution of this complex and demanding virtual phone environment.

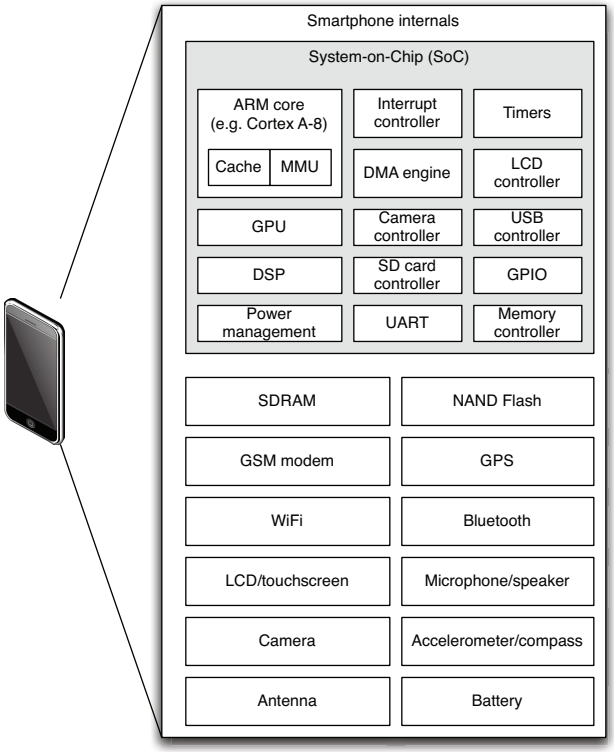


Figure 2: Smartphone components.

- **Security**. The most significant enterprise security concerns introduced by mobility relate to physical threats, for example phone theft or the removal of media containing confidential documents. The ability to dictate minimum password strengths, inactivity thresholds and lockout, storage and network encryption and remote wipe are necessary for a mobile platform to be considered secure [22].

  In addition, there are traditional software security threats that are common to desktop and mobile platforms. One of the main attractions of a smartphone is

the ability of the user to download and install applications at will. Application store distribution models popularized by Apple's iPhone App Store [4] have led to an enormous number of applications available on these markets. We are now facing the same threats from Trojans and spyware that we have seen on PCs.

In a BYOD model, phone owners should not be limited in the applications that they can download, while corporate IT needs the ability to protect corporate assets. We reconcile these opposing views by having the MVP hypervisor secure the execution and data in the enterprise virtual phone stack from the open environment on the rest of the phone.

Section 3.4 details the MVP security architecture, addressing these physical and software threats with features including an encrypted virtual filesystem and virtual private network (VPN) tunneling of corporate application network traffic.

- **Low complexity**. We strove to minimize the overall system complexity, including both that of the hypervisor *and* changes introduced in the guest to increase our confidence in the fidelity of the virtualization. Lower complexity leads to higher reliability, improved maintainability and greater security when trusted components are also kept simple.

- **Performance**. The performance of an application in the virtual phone should be indistinguishable by a human user from the performance of that application running on the native, unvirtualized device. In addition, battery life should not be unreasonably affected by the execution of the virtual phone. We wanted to go beyond the limits of microbenchmarking and optimize for the end user experience. Section 3.5 describes some of the techniques we used to measure performance and to target optimization efforts.

- **Manageability**. A key requirement for an enterprise mobile solution is the ability for IT departments to remotely manage devices, or in our case the virtual phone. This requires supporting the provisioning, updating, wiping, locking and backup of virtual phones over mobile networks.

- **OEM time-to-market (TTM)**. Ideally a mobile virtualization solution should have minimal additional engineering effort required by OEMs to support. In the next section we describe an architecture capable of achieving this by supporting the secure distribution of most hypervisor components via standard application channels. This differs from existing bare-metal hypervisor approaches in which the entire mobile hypervisor is embedded on the device by the manufacturer prior to shipping and supports a more flexible approach to hypervisor distribution.

3. SYSTEM ARCHITECTURE

3.1 Overview

Figure 3 provides a block level overview of the MVP system architecture. MVP is a Type 2, or *hosted*, hypervisor, providing the ability to execute additional *guest* virtual machines alongside a smartphone's existing *host* operating system.

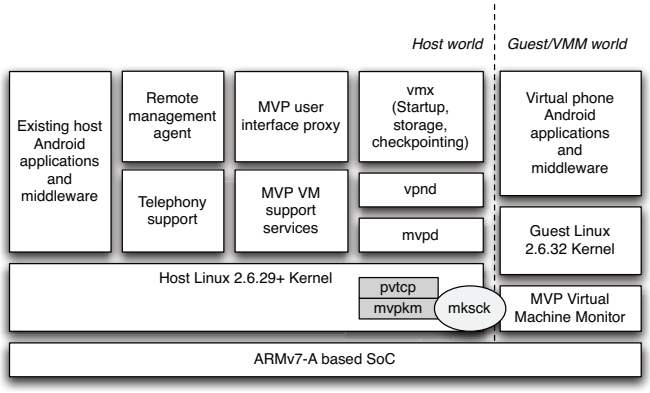


Figure 3: MVP system architecture.

Similar to the motivation given by Sugerman et al. [30] for this organization in VMware's desktop virtualization product Workstation, a hosted hypervisor provides a powerful approach in addressing the hardware diversity present in the non-core aspects of mobile device SoCs. MVP leverages the existing device drivers in the host operating system, mapping guest virtual device operations to standard interfaces exported by the host kernel. Three other key motivations led to this structure:

- The hosted model is a natural fit for BYOD. A personal phone with an existing operating system, applications and settings can be provisioned with an enterprise VM without perturbing the existing environment. Applications in the host personal phone are not affected by the presence of the enterprise VM and do not incur any virtualization overhead.

- The hypervisor need not interpose on every device in the system. For example, hardware accelerated 3D graphics necessary for games can be supported in the home environment without any explicit hypervisor awareness if not required in the enterprise environment.

- A hosted hypervisor does not have to be integrated early in the smartphone development cycle and most components can be provisioned over a mobile network.

MVP relies on the ability of certain trusted components to obtain elevated capabilities and execute in privileged modes. A minimal daemon `mvpd` executes as superuser on the host and is responsible for granting further MVP-related processes necessary capabilities. `mvpd` is the only piece that requires placement on the device by an OEM. Section 3.4 gives further details on the integrity checks that `mvpd` performs and the chain of trust that enables a verified execution environment for the VM.

`mvpd` is responsible for inserting an authenticated kernel module `mvpkm` that supports the transfer of control between the host kernel and the MVP virtual machine monitor (VMM). When the enterprise VM is launched, `mvpd` loads the guest image and VMM into memory and dedicates a thread to the execution of the VMM. From the host operating system's

point of view, this thread represents the time spent running the guest VM.

Once launched, the processor is time multiplexed between the guest/VMM and host worlds — a world being the user *and* system context on the processor, including both user and privileged register file and coprocessor state. In the host world, the existing host OS and applications continue to execute as before. When the guest VM is to be scheduled, the proxy thread in `mvpd` calls into `mvpkm` which then orchestrates the world switch. Control is transferred to the VMM which in turn passes control to the guest. The VMM retains ownership of the exception vector table and page tables, allowing it to interpose as needed. The VMM returns control to the host world on interrupts and when necessary to access host services, for example host memory allocation or to make some system call on behalf of a virtualized device.

Core virtualization is performed by the VMM and takes place entirely in the guest/VMM world — there is no need to world switch on the vast majority of traps related to instruction set or address space virtualization. The VMM works in concert with components on the host, such as the `vmx` process and `pvtcp` kernel module to present to the guest a set of virtual devices. The general model that we have adopted is to introduce a paravirtualized guest driver for each device and have the VMM intercept hypercalls for device specific behavior, forwarding requests as needed to the host components. Section 3.3 provides further details on how this applies to some of the key devices supported by MVP.

A custom message passing transport between the worlds, which we refer to later as `mksck`, is introduced to the host kernel by `mvpkm` and appears to host processes via a standard sockets abstraction. Support is provided for establishing secure shared mappings between host processes, the VMM and the guest.

3.2 Core virtualization

Almost all smartphones today are based on ARM cores, with a recent trend towards ARMv7, the latest version of the ARM architecture available in SoCs. Earlier versions of the ARM architecture featured virtually tagged and/or indexed data caches, which required careful management at the application, operating system [34] and hypervisor levels. ARMv7 has instead a simpler physically addressed data cache programming model. Given the MVP product time line, and the availability of ARMv7 support in contemporary mobile operating systems, it was advantageous to opt to support only ARMv7 since this led to a reduction in hypervisor complexity.

We describe below the approaches adopted in the MVP VMM to two aspects of core virtualization — instruction set architecture (ISA) and memory. While the lightweight paravirtualization (LPV) techniques described below are used in the production MVP VMM, the modular architecture in Section 3.1 enables other VMMs and virtualization techniques to be employed. VMware has developed internally several alternative VMM prototypes using binary translation, deprivileged user-mode virtualization [12, 29] and hardware assisted virtualization support. The last of these VMMs is based on ARM's upcoming Virtualization Extensions [10] and demonstrates the potential for eliminating guest paravirtualization with respect to the core completely.

3.2.1 ARM Instruction Set Architecture (ISA)

ARM is a 32-bit load/store architecture in which data processing instructions act only on registers. It provides a user mode for application execution and several privileged modes for kernel execution and exception handling, referred to as system mode, IRQ mode, etc. Low power variants of the ISA known as Thumb and Thumb-2 feature a subset of the instructions with a reduced instruction width and hence higher density. The architecture facilitates instruction set extension with coprocessors, used for floating-point instructions, single-instruction multiple-data (SIMD) processing instructions, etc.

Similar to other architectures such as x86, ARM has a number of non-privileged instructions (i.e. instructions that do not trap when executed from an unprivileged mode) that can potentially access or rely on privileged state. Such instructions are termed *sensitive* instructions in ISA virtualization terminology. Because of the presence of such sensitive instructions, the ARM architecture is not classically virtualizable as per the original definition proposed by Popek and Goldberg [26].

Two standard ways in which sensitive instructions can be dealt with are by using paravirtualization of the guest kernel source to replace sensitive instructions statically or by employing binary translation to catch and handle sensitive instructions dynamically. We opted for a form of paravirtualization for two reasons:

- Due to the coupling of kernel source code availability and licensing in the mobile space, paravirtualization posed no limitation on the guest kernels we could support.
- We could reduce overall system complexity, opting for simple guest kernel changes over the complexity of a complete and optimized binary translator.

Our goal in virtualizing the ISA for the guest virtual machine was to avoid *any* changes in the guest user-mode code, i.e. applications and middleware, and to make minimal changes to the hardware definition. This implied only those changes to the guest kernel that were necessary to simplify the monitor as compared to a VMM based on dynamic binary translation. The motivating reasons for minimizing changes beyond reduced complexity were the easing of guest porting to the MVP virtual platform and maintaining a stable interface between the guest and VMM.

Towards this goal, we employ the following approach to virtualize the ISA for the guest kernel:

- We execute the entire guest, both kernel and user code, in user-mode on the physical processor. We rely on privileged instruction semantics dictated by the ARM ISA to automatically trap and handle most privileged instructions encountered in the guest kernel, e.g. privileged coprocessor access instructions `mcr` and `mrc`. The

exception to this, where we replace such instructions with hypercalls, is described in the next section.

- We paravirtualize all devices, so we do not need to provide replacements for instructions that might access regions of the address space responsible for I/O as they do not exist, see Section 3.3.
- Finally, we rely on paravirtualization to handle the sensitive instructions. Most of the instructions in this class relate either directly or indirectly to state contained in the Current Processor Status Register (CPSR), which has many privileged bits of information such as the current mode of execution, whether interrupts are enabled or not, etc. However, it also has some innocuous bits of information such as the status flags, including the zero flag, the parity flag, etc. Depending on which mode of execution the CPSR register is accessed from, the semantics governing which bits are overwritten and which bits remain unmodified are different.

More specifically, we employ a variant of paravirtualization that we call *lightweight paravirtualization* (LPV). Instead of making changes to entire subsystems and data structures in the guest kernel, we limit changes to only the architecture dependent branch, e.g. `arch/arm` in the Linux kernel source tree. Further, we limit changes to surface-level changes in leaf functions/macros that can be easily made with a simple search and replace. We replace sensitive instructions either with hypercalls 1:1 to force a trap (and then emulate semantics upon trap in the monitor), or with a block of instructions that emulate the sensitive instruction's semantics in-place, a more performant approach in many cases.

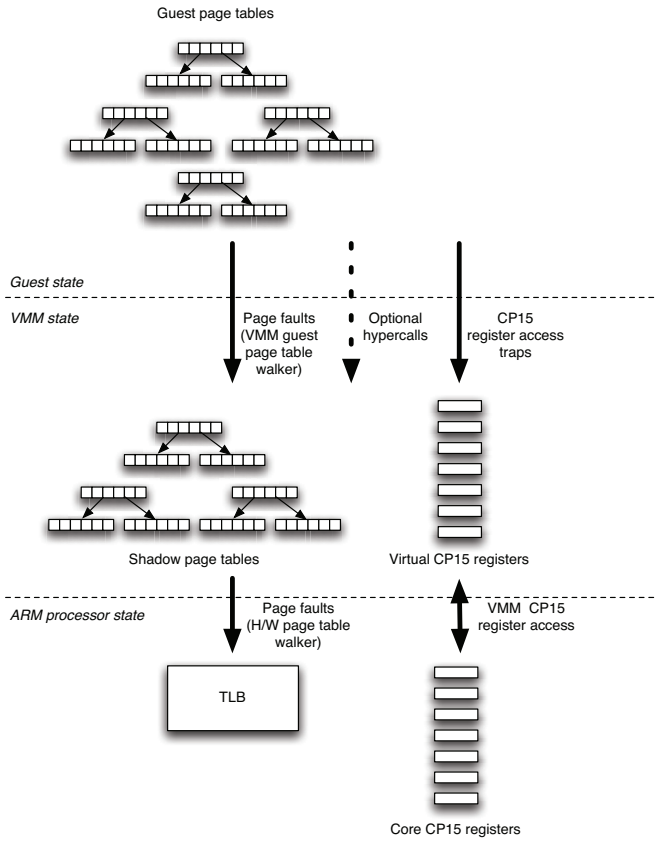


Figure 4: ARM MMU virtualization in MVP.

3.2.2 Memory

There are two components in the memory subsystem that require virtualization, the memory management unit (MMU) and the physical memory resources presented by the SDRAM on a mobile device.

MMU virtualization encompasses how the processor address translation and memory protection hardware is virtualized. The ARMv7 MMU manages a 32-bit virtual address space, represented in a two-level hardware-walked tree page table. Page sizes range from 4KB–16MB and 8-bit ASIDs are provided. In addition to the page-level read, write and no-execute permissions, a *domain* protection mechanism allows for page-level protection to be overridden at 1MB granularity [5] and modified orthogonally. The architecture supports multiple levels of TLBs, unified and split. As no dedicated hardware virtualization support exists today, i.e. no nested translation, we had the option of modifying the guest via paravirtualization or providing a virtual MMU faithful to the original architecture via trap-and-emulate. We opted for the latter since paravirtualization of page table and memory management involves making changes to extant guest data structures and algorithms, with greater complexity than a simple search and replace of sensitive instructions. Optimizations presented by paravirtualized approaches to MMU virtualization [7] can still be achieved through a set of optional hypercalls that are added to a guest kernel as needed. For example, loops of guest cache and TLB maintenance CP15 instructions are replaced with a single hypercall.

A MMU provides a virtual address space abstraction, mapping virtual addresses to physical addresses. With virtualization we introduce an additional level of indirection, with the guest maintaining as before a page table with guest virtual to guest physical mappings and the VMM maintaining a mapping from guest physical addresses to machine physical addresses. Shadow page tables [2], which cache the derived guest virtual to machine physical mappings, are maintained by the MVP VMM and are accessed by the hardware page table walker. The shadow page table contents are kept coherent with the guest page table and hypervisor physical page mapping data structure by intercepting traps resulting from guest page faults and TLB maintenance related Coprocessor 15 (CP15) register accesses. Since the guest executes in user mode, any attempt to access these privileged registers results in a trap and shadow copies of these register locations are maintained when they are stateful. Figure 4 illustrates the relationship between guest, VMM and processor state. The above traps are sufficient to faithfully virtualize the ARM MMU with shadow page tables. A number of further heuristics, caching and prefetch optimizations exist in the MVP VMM to improve performance and the end result is negligible MMU virtualization overheads for many workloads.

Machine memory is managed by the host operating system and allocated to the VMM via `mvpkm`. Since guest workloads may be demanding and require a large fraction of the device memory resources, we overcommit memory between the guest

and host, employing a ballooning [33] approach to balance the available free memory resources and page cache utilization. The balloon policy avoids the situation in which either the guest or host trigger low memory application lifecycle behavior while there remains free resources in the complementary world. MVP also supports rapid checkpointing and restore of VMs, providing an enhanced user experience via virtual phone persistence and obviating the virtual phone boot process.

3.3 Platform virtualization

Below we provide an overview of how a selection of smartphone components are virtualized in MVP. In the interest of brevity we focus on components where the smartphone hardware and software environment influenced the design choices.

3.3.1 Storage

Both hypervisor components and VM images are stored on the existing host filesystems. Smartphones typically have two types of storage devices — internal NAND Flash memory and Secure Digital (SD) cards. The internal NAND is fixed and constrained in size due to cost and power consumption. On high-end smartphones this typically ranges from 256–512MB, while SD card storage, also based on NAND technology, but connected via a SD bus and controller, provides removable storage that can extend up to 32GB today. Secure Digital Extended Capacity (SDXC) cards will support up to 2TB capacities in the future. SD card storage benefits from the economies of semiconductor scaling and supply after a smartphone has been shipped and purchased.

When the host system image, applications and data are considered, the available internal NAND storage is a fraction of the device capacity. Hypervisor components install on the internal NAND as with other host applications. Since VM images are large, containing both a tuned system image, enterprise applications and data, they lend themselves to SD card storage. The MVP storage virtualization components enable higher levels of performance, data integrity and security on the virtual device than is available on a standard SD card filesystem.

Figure 5 shows the various layers in the guest and host storage stacks. A file read or write by a guest application will require data to be transferred between guest application memory and the physical SD card media. Take for example a `write` operation on some file. Once the data has been transferred to the guest kernel and written out from its page cache, a virtual I/O operation is started by a hypercall from a paravirtualized device driver. The driver provides a reference to the monitor which, in turn, provides a shared mapping to a storage thread in the `vmx` component on the host (from Figure 3). The block is written to the VM image file using a `write` operation on the host, at which point the host I/O stack is responsible for transferring it to the physical card.

FAT-based formats are standard for SD cards and the flash translation layer (FTL) on many cards is optimized for large sequential file transfers — SD cards are primarily used for photos, music and video storage and document transfers. We provide enterprise-level performance, reliability and security with:

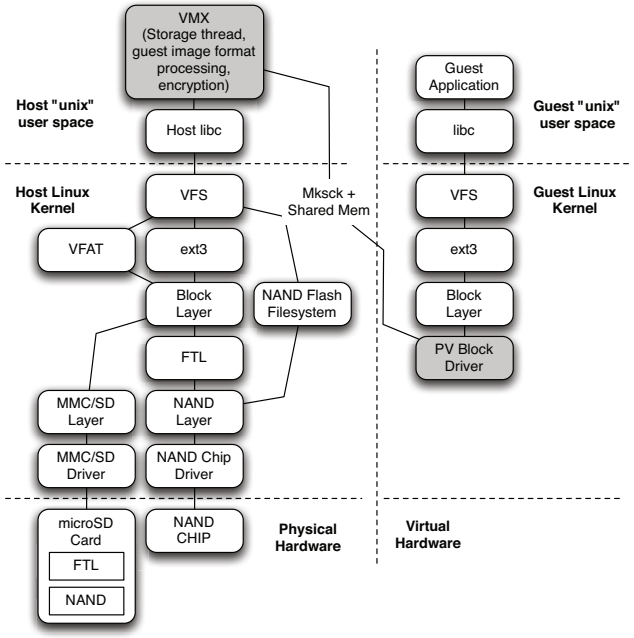


Figure 5: MVP storage architecture.

- A high-performance log structured VM image format matching the I/O mixture from the guest, containing a significant amount of small non-sequential operations, to the I/O characteristics of the SD card. Unlike their cousins found in solid state disks (SSDs), the low cost FTLs in consumer grade Flash storage devices dictate a significant penalty for non-sequential writes [3, 9]. It is not uncommon to observe two orders of magnitude or more difference between sequential and non-sequential write performance for small block sizes.

- A journaling guest filesystem. Barriers issued by the journaling EXT3 guest filesystem are propagated to the `vmx` storage thread, allowing it to take steps necessary to provide the media coherency guarantees expected by the guest. MVP bypasses the host page cache and performs direct I/O, maintaining its own buffers that are flushed in response to barrier requests. This was found to be a solution with lower maximum latency than syncing page cache contents on barriers.

- Block level encryption is used in the guest image format to mitigate threats posed by SD card removal, phone theft and malicious applications with access to the SD card.

- A mostly zero copy stack. With the exception of the encryption/decryption steps, there is no need for the CPU to directly process any guest block data between the guest kernel buffers and SD card controller DMA engine.

The checkpoint images also reside on SD card storage and have the same log structured and encrypted storage format as the guest images.

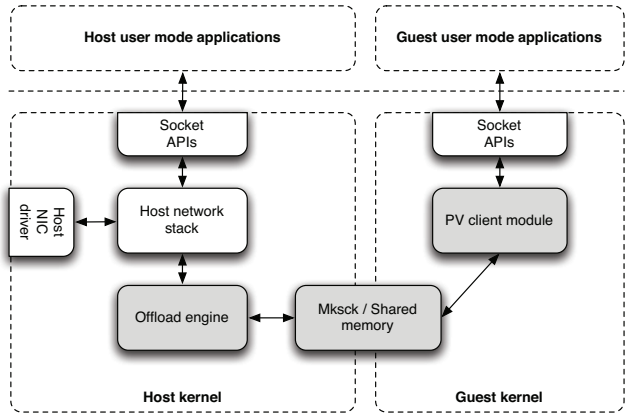


Figure 6: MVP networking architecture.

3.3.2 Network

Networking capabilities are provided by a high-level paravirtualization framework called *paravirtualized TCP* (PVTCP). PVTCP is distinguished from traditional full virtualization and paravirtualization techniques by operating at the socket/system call level as opposed to the device interface level. Using this approach allows for benefits such as lower virtualization overhead and more flexible deployment options.

The architecture of PVTCP is broadly split into two parts, a *PV client* module that runs in the guest VM kernel context, and an *offload engine* that runs in the host kernel context, implemented in the `pvtcp` module in Figure 3. In the PV client, all network services requested by guest user-mode applications are intercepted at runtime just before the transport level. These requests are then proxied off to the offload engine via the `mksck` high-speed shared-memory communication channel.

For example, when a guest VM attempts to perform a `socket` system call to open a socket, a request is made directly to the offload engine. The offload engine performs a `socket` call on behalf of the PV client, creating a proxy socket for the guest. The net effect is that a guest application attempting to execute protocol-level services will delegate protocol-processing and driver-level processing to the offload engine.

By interposing in the guest directly at the protocol level instead of at the device level, we avoid the necessity of traversing two network stacks (one for the guest and one for the host) as well as the complexity of software emulation of a hardware network interface. In fact, because all socket calls are delegated to the offload engine, the guest VM does not need to have a virtual NIC at all. This architecture allows for better CPU utilization and lower virtualization overhead. In internal testing, we have achieved throughput-parity with the host on many workloads and significantly lower CPU-utilization as opposed to a traditional fully emulated NIC approach.

All PVTCP traffic is tunneled through the `vpnd` component in the host as per the security model described in Section 3.4.

3.3.3 Telephony

It is impossible to discuss a smartphone environment without referencing the telephony components involved. For MVP, our vision is to provide isolation of personal and work environments, supporting two distinct phone numbers. MVP supports a number of approaches to realizing multiple phone numbers on a single device, with the ability to provide a specific option dependent on some subset of the device manufacturer, Subscriber Identity Module (SIM) card manufacturer, carrier network as well as enterprise network connectivity and integrated solutions. We review some of the possible approaches below and note that the actual product will support one or more of these based on market requirements and capabilities.

- **GSM.** Within the GSM space there are number of viable options. In some geographies devices capable of supporting two SIM cards are sold, although these are generally not smartphones in nature. Alternatively, a single SIM card is capable of supporting multiple lines which makes the form factor more familiar and integrated with lower device design changes. One approach is to provide multiple International Mobile Subscriber Identities (IMSIs) on a single SIM card, which allows the mobile device to appear to the network as multiple devices from a capability and billing perspective. These solutions are also sold and supported in diverse geographies. A separate approach is the alternate line service (ALS) provision in an extension to the GSM specification [1], which has been activated in some networks, although this appears to be limited in deployment and in some cases decreasing.

- **Network**. From a network perspective it is possible to offer a call-forwarding solution where numbers terminate at a PBX (e.g. an Asterisk server [6]) and are then forwarded via a third invisible line to the device with some meta-data indicating which of the original personal or work numbers were the target of the incoming call. Outgoing calls can be similarly tagged to indicate the originating number. Examples of this type of solution are common, e.g. Line 2 [23].

- **Voice-over-IP (VoIP)**. VoIP continues to be an interesting and somewhat controversial addition to the telephony options available on smartphones. As carriers begin to embrace and offer VoIP-based services, the evolution to 4G/LTE will further enhance the ability of networks to support this as a viable option. Within the MVP domain, there are two interesting approaches; firstly connecting to public domain providers like Skype or Fring. The second option which is particularly interesting in the enterprise space is the integration of Unified Communications (UC) [13] technologies and their continued penetration in the corporate data center. By merging UC and MVP, enterprises have a compelling end user experience while retaining control over policies, auditing, adding IT-controlled capabilities etc. The end-user can benefit with features like integrated dialing, easy access to instant messaging (IM) and presence tools as well as access to corporate assets like address books and accurate person availability.

3.4 Security

Below we summarize the key security threats and goals expressed by enterprises, mobile security researchers and OEMs, followed by details of the MVP approach to securing the platform.

Enterprise security goals include protecting sensitive information on the mobile device, data exchanges between the device and the corporate intranet and access to data or services within the corporate intranet. As discussed in Section 2, an additional goal is to counter the physical threat model posed by devices that may be lost or stolen. To achieve these goals today requires that corporate IT policy place inconvenient restrictions on the installation of applications, password configurations and device lockout.

A significant threat concerning the security community [8, 14, 15] and OEMs is posed by the untrusted downloadable applications that may exist in the host environment. Such applications may have information flows beyond those anticipated by the user and/or may be actively malicious, e.g. Trojans, spyware and other malware. On the Android platform, the end user is prompted when an application is installed to confirm the permissions granted to access devices such as the GPS, SD card and various OS services. It is common for applications to request permissions that are not obviously connected with the application's function and end users may not have the required technical understanding to effectively arbitrate, e.g. it may be difficult to understand why a game requires access to the address book, network and GPS location, and whether this request is reasonable. This confusion can lead to Trojans bypassing the manual permission checks. According to a recent report [31], 20% of applications in the Android market request permissions to access private or sensitive information and 5% have the ability to place a call with no user check. Another illustrative example of a malicious host application is the introduction of a proxy application linking an adversarial external host and a corporate intranet [11].

MVP secures corporate resources accessible by employee owned devices by creating two separate environments. The host environment is open and the device owner can continue to use it without restriction. The enterprise environment is managed by corporate IT, which decides exactly what kind of application can be installed in this environment and which policies must be applied. In addition:

- Irrespective of the permissions given to applications in the personal environment, they cannot cross the virtualization barrier, which prohibits host applications gaining access to the network, address book, SMS, phone dialing and other data/resources in the enterprise VM.
- All network traffic from the enterprise VPN is tunneled through a VPN to a corporate VPN gateway via a daemon `vpnd`, shown in Figure 7. Applications in the enterprise VM transparently benefit from the VPN access. The host kernel `tun` device and `pvtcp` share a namespace isolated from the normal network namespace on the host (a feature available in the Linux kernel since version 2.6.24). This prevents untrusted host Android applications from gaining VPN access, ruling out the possibility of a malicious application bridging an external host to the VPN.
- There are only very limited access controls on the SD card in Android. All files are accessible from any application with SD card permission. The card is also removable by anyone with physical access to the device. As described in Section 3.3.1, the MVP solution implements a block level encryption of the guest's virtual file system and checkpoint images to mitigate these risks. Thus, all applications in the enterprise VM benefit from storage encryption and are protected from prying host applications and SD card removal. This is illustrated in Figure 7.
- A password is required to start the enterprise VM and depending on corporate policy, to switch from personal to work environment, either each time, or when the work VM has been inactive for some period of time.

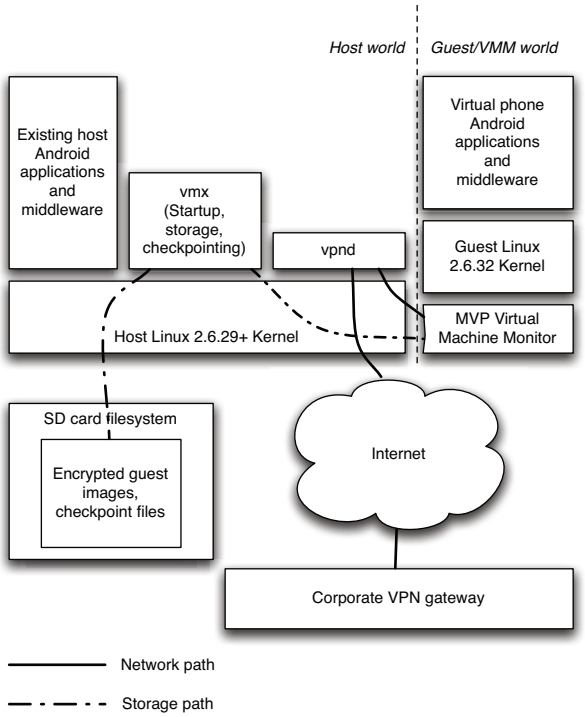


Figure 7: MVP storage and networking paths.

From an architecture point of view, the MVP solution has been split into different components, implementing a privilege separation pattern.

When performing privileged operations on the host, the root of trust is always the `mvpd` process. It is installed in a read-only filesystem by the OEMs, effectively enabling virtualization on the phone. In conjunction with Android's signing and permission model [28], `mvpd` authenticates the other trusted and, transitively, the untrusted components. The trusted components are kept simple and small. The other virtualization components on the host, which are used

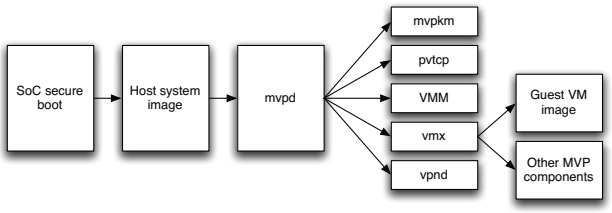


Figure 8: MVP chain of trust. Arrows indicate that the source authenticates the destination node.

to implement device virtualization or platform management, run with user privileges. For example, the applications responsible for the host side of display or audio virtualization don't need to be run as the superuser. In the guest/VMM world, only the VMM component executes in a privileged mode.

At boot time, a chain of trust is created, preventing image tampering, as shown in Figure 8. The phone boot code authenticates the phone system image. `mvpd` is authenticated as part of this system image. Before starting the work VM, `vmx` authenticates the other virtualization components installed on the platform, as well as the VM system or checkpoint image.

3.5 Performance methodology

MVP's BYOD use case focused our performance efforts on end-user experience with no hard real-time constraints. Our goal was that the performance of an application in the virtual phone should be indistinguishable by a human user from the performance of that application running on the native, unvirtualized device. Our approach to track progress toward this goal involves continuous benchmarking with workloads that include, among others:

- **Web browsing**: a scripted test that measures the time to start a browser, load a page with several images, and scroll down that page.
- **Bootup**: starting an Android-based virtual machine to stress storage, the Android JVM and process creation.
- **Multimedia kernels**: predominantly unprivileged, easy-to-virtualize computation that serves mostly as a practical sanity check that virtualization imposes little overhead with such programs.
- **OS and I/O microbenchmarks**: used by subsystem developers to isolate and improve a particular aspect of the system.

While we strove to avoid virtualization overhead in all workloads, it was a guiding philosophy that we would tolerate even substantial overheads if they could be shown to have no effect on an interactive user's experience and battery lifetime.

To a first order, battery life is proportional to execution time and the active cycles of the display and modem hardware. Hence, a system with low virtualization performance overhead can be expected to have low energy overhead, providing that host device management is unaffected by the presence of a hypervisor. However, special care must be taken to allow the phone to take advantage of any low-power states provided by its hardware. To this end, certain polling-based subsystems have been replaced with on-demand implementations. We study power consumption of an idle MVP system using a battery simulator and proxies such as host OS-reported scheduler wake-ups per second.

We describe below the performance measurement and analysis methodology for the VMM and host components of the system, together with an overview of how we manage performance regressions.

3.5.1 VMM

Our primary tool for diagnosing and correcting performance problems is an internal tool called `kstats`. This tool was originally developed for the VMware x86 VMM to provide lightweight, flat execution profiles of VMM *services*. Call chains involving the VMM do not employ a traditional stack: while the VMM code makes function calls to VMM routines, the VMM is preemptible and may be entered or exited via exceptions and world switches. This program flow makes function-level profiling difficult. Furthermore, a function (like `HandleInterrupt`) may be called on behalf of various services (emulating a device, servicing a page fault, etc...), and the key to improving performance may have more to do with reducing the number of times such a service is required than speeding up a leaf function it invokes. The trade-off for achieving such service-oriented profiling is that `kstats` requires developers to annotate the VMM code. As the annotations are simple, the manual effort is an acceptable tradeoff given the importance of VMM performance.

An example of a `kstats` profile is shown in Figure 9. The profile is an example of a workload dominated by easy-to-virtualize unprivileged code with little I/O. The name of the service appears at the left. The next column shows the percentage of total execution time attributed to each service. The Average Cycles column is merely the quotient of Total Cycles divided by Counts. The Overhead column is an estimate of virtualization overhead computed with the formula: `Total Cycles/(de.User + de.Priv)`. We assume that the workload running without virtualization would require approximately the number of cycles spent in direct execution. Thus, the ratio of a VMM service's cycles to the estimate of native execution cycles represents the overhead caused by that service. Note that the overheads need not sum to 1.

Looking at both the Overhead and Percent column gives developers two views of the same profile: a view sorted by Overhead shows the most expensive services while one sorted by Percent puts these services in the context of the entire execution. One might be drawn to optimize a service with a large overhead, but this may be a low-priority task if that service does not contribute much to the overall slowdown. Alternatively, a service may not currently be the largest component by percent of total time, but if it has a high overhead, its effect may become more pronounced as the impact of other services is reduced.

In addition to a text-based cumulative profile, it can be useful to visualize the profile of a workload over time. An interactive, graphical tool has been developed for this purpose, and a

```
-------------------------------------------------------------------------------------------------------
Name                                                    %        Total     Counts    Average    Ovrhd
                                                                 Cycles               Cycles
-------------------------------------------------------------------------------------------------------
de.User (Direct Execution of guest user mode code)   91.759%   8809370286     2076    4243434       -
host    (Host time: vmx and non-MVP processes)        3.749%    359887170     5474      65744    0.04
de.Priv (Direct Execution of guest privileged code)   1.438%    138087476     2077      66484       -
irq     (Handling interrupt)                          0.465%     44663350     2266      19710    0.00
swi.PrivToUser (Guest mode switch)                    0.412%     39565001     2076      19058    0.00
...
```

Figure 9: The first few lines of a kstats profile of an unprivileged multimedia workload

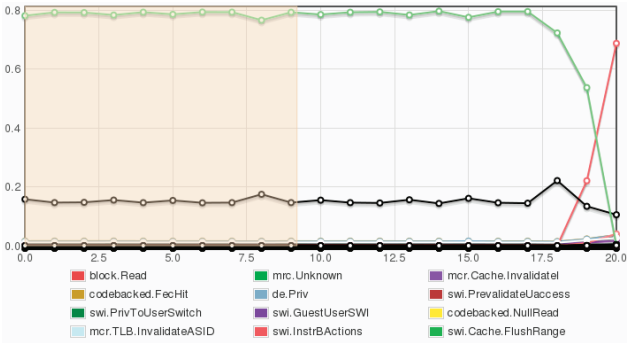


Figure 10: Time series view of a `kstats` profile. The legend has been truncated for brevity.

screen shot is shown in Figure 10. Such a view allows the user to restrict analysis to the steady state behavior or a particular program phase. The `kstats` tool suite includes other visualizations that enable MVP engineers to focus optimizations.

3.5.2 Host

`kstats` only provides fine grained information about time spent in the VMM. The rest of execution consists of direct execution of the guest or execution in the host world. Host world time appears in a `kstats` profile, but the asynchronous nature of the host MVP components and batching of tasks that they perform for the VMM make it difficult to interpret the information; it is included mostly as a placeholder. To understand how time is spent on the host, we use `oprofile` [24]. `oprofile` compatibility is achieved via careful transitions at each world switch. When execution transitions from the host to the VMM, the MVP world switch code saves the values of the performance counters and their status registers. Prior to resuming execution on the host, the values are restored, giving the appearance that no time has elapsed and no counter has overflowed due to VMM or guest execution.

3.5.3 ARM Performance Monitoring Hardware

We also take advantage of the multiple hardware performance counters available on our platform. These can be useful for precise timing of a code routine (e.g., by performing that routine in a tight loop surrounded by reads of the cycle counter). We have measured cache and TLB miss events to determine whether code that shows up as a `kstats` hotspot would benefit from improving the data layout or reducing the code footprint.

3.5.4 Regressions

In addition to low-level profiling to reveal and correct particular problems, the MVP team runs continuous tests of performance to track progress and catch regressions. The results of this testing are recorded in a database and plotted on a website that provides links to the associated log files, profiles and changesets. When the continuous integration system or an interactive user detects a result that is significantly different from the baseline in the database, we can consult the associated artifacts to determine if the change is due to inherent variability or the corresponding code change.

4. VIRTUALIZING ANDROID — MVP IN ACTION

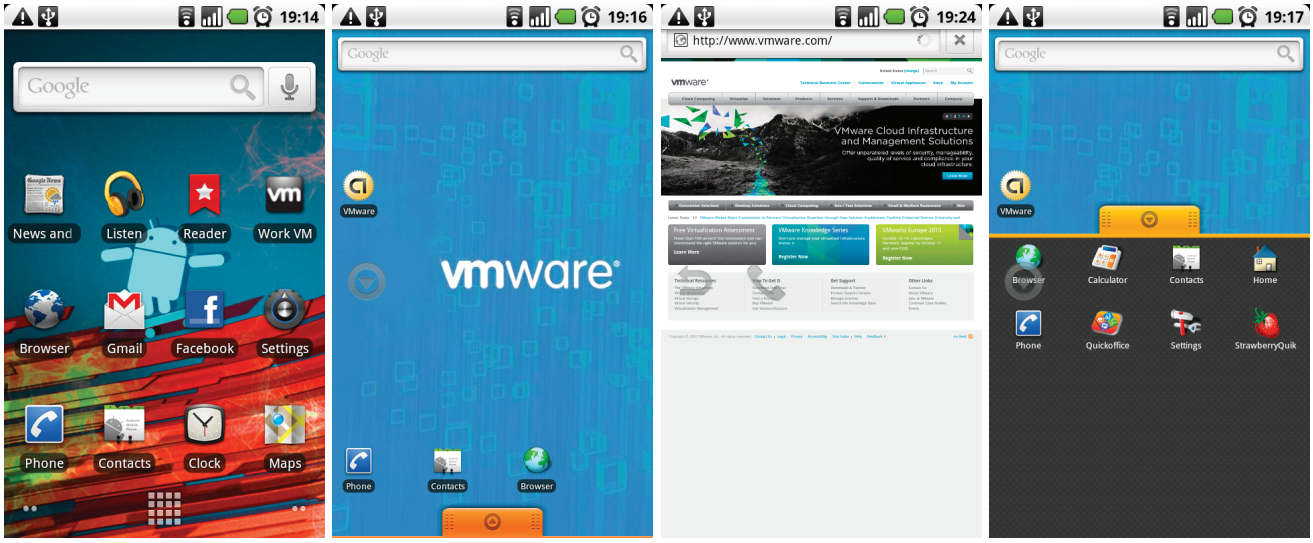
Figure 11 provides a flavor of the user experience when interacting with the dual personal/enterprise phone environments provided by MVP. The screenshots were obtained from a Nexus One smartphone with both Android 2.1 guest and host. Even though the underlying mobile OS in both environments is similar in this example, there is distinct isolation provided, both at the system level as discussed in earlier sections and through the user interface (UI).

The home screen of the personal phone is shown in Figure 11(a). Various outside-of-work applications are installed, such as Facebook, GMail and a podcast aggregator. Also, an icon exists in the top right corner for the enterprise VM. The VM may be running in the background or checkpointed. Once this icon is tapped, the UI switches to that of the enterprise VM, as seen in Figure 11(b). Different, IT provisioned, applications are installed and the Android instance has an alternative theme. The browser in Figure 11(c) connects via the corporate VPN to the network. By selecting the "Home" icon in Figure 11(d), the user can switch control back to the personal phone environment.

Further demonstrations of MVP are available at:
`http://www.vmware.com/mobile`.

5. RELATED WORK

In this paper we describe the architecture of a hosted hypervisor for mobile devices based on the ARM architecture, aimed at supporting an enterprise BYOD model. Similarities exist in the system structure with VMware's Workstation hosted architecture for x86 PCs, described by Sugerman et al. [30], however there are significant differences in the host world componetization, influenced by the mobile network provisioning model for the hypervisor.



(a) Personal phone home. (b) Enterprise VM home. (c) Browsing inside VM. (d) Return to personal phone

Figure 11: MVP personal/enterprise phone screenshots.

Early commercial use cases for mobile virtualization were typically oriented around the concerns of silicon vendors, for example consolidating application and baseband processors or providing GPL isolation [35]. Type 1, or *bare-metal*, hypervisor architectures [20] were common, offering real-time advantages [32]. The BYOD enterprise use case we address with MVP invokes a different set of requirements, described in Section 2, making an alternative approach to hypervisor structure and provisioning compelling.

To the best of our knowledge, all previous approaches to system virtualization on the ARMv4–7 architectures have entailed some form of core paravirtualization, for example Xen on ARM [21]. MVP employs a distinct shallow paravirtualization approach described in Section 3.2, requiring only the identification and replacement of sensitive instructions. At the application level, Sehr et al. [27] describe techniques for sandboxing on ARM via rule constraints on binaries and static validation, allowing for the co-existence of trusted and untrusted components in the same address space. Application performance measurement by dynamic binary translation on ARM is described by Hazelwood and Klauser [19]. VMware has also developed a dynamic binary translation based research VMM for the ARM ISA that eliminates the need for any paravirtualization. The close coupling of licensing and source code availability in the mobile domain allows for an overall system complexity reduction by lightweight paravirtualization.

6. SUMMARY

In this paper we've presented the underlying motivation for smartphone virtualization, the BYOD use case, and provided a tour of the MVP hypervisor's design goals and system structure, including the security model and performance methodology employed. We believe that the technical architecture underpinning the MVP hypervisor provides a robust, secure and performant approach to provisioning a device with both personal and enterprise managed environments, while providing a portable solution to OEMs with low time-to-market.

With MVP we feel that we can adequately address the growing market need to support employee-owned smartphones in the modern enterprise. Finding the right balance of IT-driven control, end-user experience, performance, security and scalability is a constant struggle in this environment and with MVP we can meet these challenges across all components of the solution. We look forward to working with our partners across the various industries to bring the product to market and are excited by the possibilities it provides here and in adjacent and orthogonal markets.

Acknowledgments. In addition to the paper authors, there are many MVP team members who have contributed profoundly to the design and implementation of the hypervisor and our overall product solution. We are indebted to Scott Devine, Larry Rudolph and Julia Austin for providing the conception and founding vision for MVP, together with the hosted MVP structure.

7. REFERENCES

[1] 3GPP, Orange PCS Ltd, Mercury One-2-One. *Common PCN Handset Specification*, 2000. Version 4.2.

[2] Adams, K., and Agesen, O. A comparison of software and hardware techniques for x86 virtualization. In *ASPLOS-XII: Proceedings of the 12th international conference on Architectural support for programming languages and operating systems* (New York, NY, USA, 2006), ACM, pp. 2–13.

[3] Agrawal, N., Prabhakaran, V., Wobber, T., Davis, J. D., Manasse, M., and Panigrahy, R. Design tradeoffs for SSD performance. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference* (Berkeley, CA, USA, 2008), USENIX Association, pp. 57–70.

[4] Apple Inc. Apple's App Store downloads top three billion. http://www.apple.com/pr/library/2010/01/05appstore.html, Jan. 2010.

[5] ARM Limited. *ARM Architecture Reference Manual: ARMv7-A and ARMv7-R edition*, 2007. ARM DDI 0406A.

[6] Asterisk — the open source telephony projects. http://www.asterisk.org.

[7] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., and Warfield, A. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM, pp. 164–177.

[8] Bickford, J., O'Hare, R., Baliga, A., Ganapathy, V., and Iftode, L. Rootkits on smart phones: Attacks, implications and opportunities. In *HotMobile'10: Proceedings of the 11th Workshop on Mobile Computing Systems and Applications* (Annapolis, Maryland, USA, February 2010), ACM Press, New York, NY, USA, pp. 49–54. http://doi.acm.org/10.1145/1734583.1734596.

[9] Birrell, A., Isard, M., Thacker, C., and Wobber, T. A design for high-performance flash disks. *SIGOPS Oper. Syst. Rev. 41*, 2 (2007), 88–93.

[10] Brash, D. Extensions to the ARMv7-A architecture. In *Hot Chips 22* (Stanford University, California, Aug. 2010).

[11] D'Aguanno, J. Blackjacking — 0wning the enterprise via the Blackberry. In *Defcon 14* (Las Vegas, NV, August 2006).

[12] Dike, J. A user-mode port of the Linux kernel. In *ALS'00: Proceedings of the 4th annual Linux Showcase & Conference* (Atlanta, Georgia, 2000), USENIX Association.

[13] Elliot, B., and Blood, S. Magic quadrant for Unified Communications. *Gartner RAS Core Research Note*, G00201349 (2010).

[14] Enck, W., Gilbert, P., gon Chun, B., Cox, L. P., Jung, J., McDaniel, P., and Sheth, A. N. TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (Oct. 2010).

[15] Enck, W., Ongtang, M., and McDaniel, P. On lightweight mobile phone application certification. In *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security* (New York, NY, USA, 2009), ACM, pp. 235–245.

[16] Forrester Research, Inc. Collaboration needs will fuel a smartphone surge, Jan. 2010.

[17] Gartner, Inc. Gartner says Android to become no. 2 worldwide mobile operating system in 2010 and challenge Symbian for no. 1 position by 2014. http://www.gartner.com/it/page.jsp?id=1434613, Sept. 2010.

[18] Google Inc. Android dev guide: What is android? http://developer.android.com/guide/basics/what-is-android.html.

[19] Hazelwood, K., and Klauser, A. A dynamic binary instrumentation engine for the ARM architecture. In *CASES '06: Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems* (New York, NY, USA, 2006), ACM, pp. 261–270.

[20] Heiser, G., and Leslie, B. The OKL4 Microvisor: Convergence point of microkernels and hypervisors. In *Proceedings of the 1st Asia-Pacific Workshop on Systems* (New Delhi, India, Aug 2010).

[21] Hwang, J., Suh, S., Heo, S., Park, C., Ryu, J., Park, S., and Kim, C. Xen on ARM: System virtualization using Xen hypervisor for ARM-based secure mobile phones. In *Proceedings of the 5th Annual IEEE Consumer Communications and Networking Conference* (Las Vegas, NV, Jan. 2008), pp. 257–261.

[22] Kaneshige, T. 7 steps to stronger enterprise iPhone security. *CIO Magazine* (Aug. 2010).

[23] Line 2. http://www.line2.com.

[24] OProfile. http://oprofile.sourceforge.net.

[25] PDAdb.net. http://pdadb.net.

[26] Popek, G. J., and Goldberg, R. P. Formal requirements for virtualizable third generation architectures. In *SOSP '73: Proceedings of the fourth ACM symposium on Operating system principles* (New York, NY, USA, October 1973), vol. 7, ACM Press.

[27] Sehr, D., Muth, R., Biffle, C. L., Khimenko, V., Pasko, E., Yee, B., Schimpf, K., and Chen, B. Adapting software fault isolation to contemporary CPU architectures. In *Proceedings of the 19th USENIX Security Symposium* (2010), pp. 1–11.

[28] Shabtai, A., Fledel, Y., Kanonov, U., Elovici, Y., Dolev, S., and Glezer, C. Google Android: A comprehensive security assessment. *IEEE Security and Privacy 8* (2010), 35–44.

[29] Steinberg, U. Fiasco μ-kernel user-mode port. TU Dresden, Dec. 2002.

[30] Sugerman, J., Venkitachalam, G., and Lim, B.-H. Virtualizing I/O devices on VMware Workstation's hosted virtual machine monitor. In *Proceedings of the General Track: 2002 USENIX Annual Technical Conference* (Berkeley, CA, USA, 2001), USENIX Association, pp. 1–14.

[31] Vennon, T., and Stroop, D. Threat analysis of the Android Market. Tech. rep., SMboile Systems, 2010.

[32] Mobile handset design: Realizing flexible, low cost, higher security, device management using OpenOS and real-time virtualization^{TM}. Tech. Rep. TR-06-102.1, VirtualLogix Inc., 2006.

[33] Waldspurger, C. Memory resource management in VMware ESX Server. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation* (December 2002).

[34] Wiggins, A., Tuch, H., Uhlig, V., and Heiser, G. Implementation of fast address-space switching and TLB sharing on the StrongARM processor. In *Proceedings of the 8th Asia-Pacific Computer Systems Architecture Conference* (Aizu-Wakamatsu City, Japan, Sep 2003), Springer Verlag.

[35] Zoppis, B. Using a hypervisor to reconcile GPL and proprietary embedded code. *LinuxDevices.com* (Aug. 2007).